# Exercises: ASP.NET Core – Watchlist, Identity and Security

Problems for exercises for the ["ASP.NET Core Fundamentals" course @ SoftUni](https://softuni.bg/trainings/4953/asp-net-fundamentals-may-2025)

A movie ticket and popcorn

Description automatically generated

## Introduction

In this workshop, we will take the CinemaApp project to the next level by introducing **user interaction** and **personalization**.

Until now, every user had access to the same movie data. Now we will allow **authenticated users** to manage their own personal **Watchlist** — a list of movies they wish to watch later.

In addition, we will:

* Implement the necessary **UserMovie** entity to support the Watchlist feature
* Apply **Authorization** rules to protect certain actions in the app
* Add basic **security measures**

At the end of this workshop, our application will be able to:

* Distinguish between **logged-in** and **anonymous** users
* Allow users to **maintain their personal Watchlist**
* **Protect** Add, Edit, and Delete **functionalities** from **unauthorized access**

## IdentityUser – Movie Relation

In order to implement the Watchlist feature, we need a way to **connect users to the movies** they have added to their Watchlist.

### Creating the Data Model

This is a classic **many-to-many relationship**:

* **One user** can add **many movies** to their Watchlist
* **One movie** can appear in the Watchlists of **many users**

To model this relationship in our database, we will create a new table called **UserMovie**, which will serve as a **mapping table** between IdentityUser and Movie.

*At this stage, we are still using the default* IdentityUser *class for user identity management. Therefore, our UserMovie model will reference* IdentityUser*. In a future workshop, when we extend Identity, this model will be easily adaptable to work with* ApplicationUser*.*

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

### Adding the UserMovie DbSet in DbContext

This ensures that Entity Framework Core knows about the UserMovie entity and can create the corresponding table.

A screen shot of a computer program

AI-generated content may be incorrect.

### Create a Configuration File

Next, we will configure the **composite primary key** and the relationships by using a clean   
**Fluent API configuration file**.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

### Understanding the Fluent API

In this project, **we are using the Fluent API to configure the relationships between our entities**.

* The **Fluent API allows us to use C# code to describe how Entity Framework Core should map our entity classes** to database tables.
* This is an **alternative to using only data annotations** (like **[Key]**, **[ForeignKey]**, **[Required] ... etc.**) inside the entity classes.

### Elements of the Fluent API in Our UserMovieConfiguration

* HasKey() — Composite Primary Key  
  The primary key of the UserMovie table is made up of two columns: UserId and MovieId.
* **HasOne() → WithMany() → HasForeignKey()** — Relationship to User and Movie
* OnDelete(DeleteBehavior.Cascade)
  + If a **user is deleted** → **all related Watchlist entries will also be deleted** automatically
  + If a **movie is deleted** → **all related Watchlist entries will be deleted** automatically

### Apply Migration and Update the Database

Now that we have fully configured the UserMovie entity, it is time to apply this change to the database.

In the previous subsection, we wrote a clean Fluent API configuration in the UserMovieConfiguration class. We also applied this configuration in the OnModelCreating method of the CinemaAppDbContext. The next step is to **generate a new EF Core migration** so that Entity Framework can **update our actual database schema**.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

When this command completes, **Visual Studio will generate a new migration class under the Migrations folder** in your project. The migration will contain the instructions needed to create the UsersMovies table.

The next step is to **update the database** with this migration.

A screenshot of a computer program

AI-generated content may be incorrect.

After this command finishes successfully, the database will be updated. You can verify this by opening **SQL Server Management Studio** (SSMS) or the database explorer inside Visual Studio.

A screenshot of a computer

AI-generated content may be incorrect.

## Watchlist

In this part of the workshop, we will begin implementing the **Watchlist feature**, which will allow logged-in users to maintain their own list of movies they want to watch later.

We will start by creating the Watchlist page itself. This is a simple Razor view that will display a list of movies the user has added to their Watchlist.

### Building the Watchlist View



Now that we are preparing to implement the Watchlist feature, the first step is to create a dedicated **Watchlist view**, where each user will be able to see their personal list of saved movies.

* To follow the structure of an ASP.NET Core MVC application, we will add a new folder inside the   
  Views folder -> Watchlist

A screenshot of a computer

AI-generated content may be incorrect.

This **folder will hold all the Razor views** related to the Watchlist feature.  
For now, we will start with a single view: **Index.cshtml**, which will display the user’s Watchlist.

If you try to access the Watchlist page right now by going to:

**https://localhost:port/Watchlist/Index**

You will receive a **404 error** saying that the page was not found:

A computer screen shot of a computer screen

AI-generated content may be incorrect.

This is normal. The reason is simple: we have only created the **Razor view** so far (Index.cshtml), but we have not yet created the **WatchlistController, but before that... ViewModel**

### Creating the WatchlistViewModel



In the case of the Watchlist page, we want to **display a list of movies** that the user has saved to **their Watchlist**.

For each movie, we want to show the following information:

* Movie poster (**ImageUrl**)
* **Title**
* **Genre**
* **Release date**
* A unique **Movie ID** (used for links and the Remove button)

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

### Creating the WatchlistController

Now that we have prepared the Watchlist view, it is time to create the corresponding **WatchlistController**.  
The controller will handle all actions related to the Watchlist feature.

At this point, the WatchlistController will include the following actions:

* Index → display the user’s Watchlist
* Add → add a movie to the Watchlist
* Remove → remove a movie from the Watchlist

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

For now, we will start by creating the controller with just an empty Index action, so we can test that routing is working correctly.

### Testing the Controller

This time, the **Watchlist page should open correctly** — you should no longer get a 404 error.  
Of course, the page will still be empty because we haven’t implemented the service yet.

A screenshot of a computer

AI-generated content may be incorrect.

### Implementing the WatchlistService and Connecting the Controller

Now that the WatchlistController is created, we are ready to implement the **WatchlistService**, which will provide the data for the Watchlist page and handle adding and removing movies.

Following our project architecture, we will create:

* An interface IWatchlistService — inside the CinemaApp.Services.Core.Interfaces folder
* A concrete implementation WatchlistService — inside the CinemaApp.Services.Core folder

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

## Authentication, Authorization, and Security Requirements

Before we continue with implementing the Add and Remove functionality for the Watchlist, it is important to clearly define the Authentication, Authorization, and Security Requirements for this part of the application.

This ensures that we implement the correct restrictions and behaviors when users interact with the Watchlist and other pages of the app

### Identity Requirements

For this version of the application:

* We are using the **default IdentityUser class**
* We will scaffold Identity (if not already scaffolded)
* We will configure Identity with the following password requirements:
  + Require confirmed account → false
  + Require digits → false
  + Require non-alphanumeric characters → false
  + Require uppercase letters → false
  + Require lowercase letters → false

### Page Access Requirements

#### Guest (not logged-in) users:

* **Can Access**:
  + Home Page
  + Movies Index Page
  + Movies Details Page
  + Login Page
  + Register Page
* **Cannot Access**:
  + Watchlist Page (/Watchlist/Index)
  + Watchlist Add and Remove functionality
  + Movie Create, Edit, Delete functionality

A close up of a screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

If a guest **attempts to access these pages or actions directly** (for example, by typing the URL), the application must **redirect them to the Login page.**

#### Logged-in users:

* **Can access**:
  + Movies Index Page
  + Movie Details Page
  + Watchlist Page
  + Watchlist Add and Remove functionality
  + Movie Create, Edit, Delete functionality
  + Logout functionality

#### Watchlist Page Behavior

* Accessible only for **logged-in users**
* If the Watchlist is empty, display a friendly message
* The page will display cards for each movie in the user's Watchlist
* Each card will have a "**View Details**" button and a "**Remove**" button
* The "**Remove**" button will **POST** to the WatchlistController → **Remove** action

A screen shot of a movie theater

AI-generated content may be incorrect.

A screen shot of a movie theater

AI-generated content may be incorrect.

#### Watchlist Button Behavior:

Movie card, **Index** page and Movie **Details** page:

* If the user is **logged in**:
  + The "**Add to Watchlist**" button is visible.
  + If the movie is **already in the user's Watchlist**, the **button is hidden** (to prevent duplicates)
* If the user is **not logged in**:
  + The "**Add to Watchlist**" button is **not visible**.

A close up of a screen

AI-generated content may be incorrect.

#### Redirection Behavior:

* After successful Login → redirect to /Home/Index
* After adding a movie to the Watchlist → redirect to /Movie/Index
* After removing a movie from the Watchlist → redirect to /Watchlist/Index
* After successful Logout → redirect to the Home page

#### Security and Code Quality Requirements:

* All **WatchlistController actions** must be protected with **[Authorize]**
* Guests should not be able to perform Add or Remove actions on the Watchlist
* Input validation:
  + The Movie ID passed to Add and Remove actions must be validated
  + We must ensure the movie exists
  + We must ensure the user can only manipulate **their own Watchlist**
* Maintain clean architecture and code quality throughout the Watchlist feature

#### Ownership and Action Permissions:

* Watchlist:
  + Users can only see and manage **their own Watchlist**
  + A user **cannot add the same movie to their Watchlist more than once**
  + Users can only remove movies from **their own Watchlist**
* Movies:
  + Only the user who **added a movie** can:
    - Edit that movie
    - Delete that movie
  + If a movie was added by another user:
    - Other users can only view the Movie Details page
    - They can add the movie to their own Watchlist if desired
    - They cannot see the Edit or Delete buttons for that movie

This **ownership restriction must be enforced both in the UI and in the controller actions**

#### URL Protection and Anti-Tampering:

The application **must not rely only on the UI to enforce security**.

Even if a malicious user tries to access restricted actions by manipulating the URL (for example, by entering /Movie/Edit/{id} or /Watchlist/Remove?movieId={id})

* The controller must validate:
  + That the current user is authorized to perform the requested action.
  + That the current user owns the resource they are trying to edit or delete.
  + That the movie exists.
  + That the movie belongs to the user (if editing or deleting), or that the movie is in the user’s Watchlist (if removing from Watchlist).

If **the user is not authorized**, the action must return Unauthorized() or Forbid() and should never proceed.

## Securing the Application – Creating the BaseController and Setting Authorization

Before we implement the Add and Remove functionality of the Watchlist, we need to first secure the entire application.

It is important to understand that **security must not depend only on UI buttons**.  
The controller actions themselves must enforce authorization, so that even if a user tries to manipulate the URL, they will not be able to access restricted actions.

### Base Controller Class

We will apply **Authorization globally** first:

* We will create a BaseController class in the Controllers folder
* All our other controllers will **inherit from this BaseController**
* We will decorate the BaseController with the [Authorize] attribute. This will make authorization **active by default for all actions in all controllers**
* We will then explicitly add [AllowAnonymous] to actions or entire controllers that should be public

A screenshot of a computer screen

AI-generated content may be incorrect.







### [AllowAnonymous] Where Needed

Now we will explicitly allow anonymous access to selected actions:

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

## IWatchlistService

A computer screen shot of a program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

## WatchlistService Methods

### GetUserWatchlistAsync()

A screen shot of a computer program

AI-generated content may be incorrect.

### IsMovieInWatchlistAsync()A screen shot of a black background AI-generated content may be incorrect.

### AddToWatchlistAsync()

A screen shot of a computer program

AI-generated content may be incorrect.

### RemoveFromWatchlistAsync()

A screen shot of a computer program

AI-generated content may be incorrect.

## Helper Methods in BaseController

### IsUserAuthenticated()

To make our controllers cleaner and more readable, we will now add a small helper method to our BaseController.

This method will allow us to easily **check whether the current user is authenticated**.

Even though we have already applied [Authorize] to our controllers, **it is still useful to have an explicit method** for cases where we want to perform **additional checks** inside an action, or when writing actions that allow both anonymous and authenticated access but behave differently depending on the **user state**.

A screen shot of a computer program

AI-generated content may be incorrect.

### GetUserId()

This method will return the current user's unique ID (ClaimTypes.NameIdentifier)

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

We will now use these methods in all actions that need to retrieve or manipulate data for the current user.

This is a small but very good practice that improves code clarity and reduces repetition

## Disable the [Add to Watchlist] Button for Guest Users

A black and white rectangular object

AI-generated content may be incorrect.

**Hiding the "Add to Watchlist" Button for Guests**

In the Movies Index page, the "**Add to Watchlist**" button must be visible only for logged-in users.

To achieve this, we will use a simple Razor condition:

A screen shot of a computer program

AI-generated content may be incorrect.

A close up of a screen

AI-generated content may be incorrect.

A close up of a screen

AI-generated content may be incorrect.

## Editing the Movies Index View → Add to Watchlist Button



* We will use a **POST form**, because adding a movie is a modifying action
* We will display the button **only for logged-in users**

A screen shot of a computer program

AI-generated content may be incorrect.

## WatchlistController Actions

Now that we have implemented the WatchlistService methods, we will update the WatchlistController to call these methods and complete the Watchlist feature.

A screen shot of a computer program

AI-generated content may be incorrect.

As we proceed, we must **carefully follow the security requirements** we defined earlier:

* Only logged-in users can access **Watchlist actions** → **enforced globally** by BaseController + Authorize
* A user can only manage **their own Watchlist**
* We must validate all input in the controller
* If a **user tries to tamper with the URL**, our **controller must defend** against this

### Index Action

Now that we have added our two helper methods — IsUserAuthenticated() and GetUserId() — we can write the Index action of the WatchlistController in a very clean and professional way:

A screen shot of a computer program

AI-generated content may be incorrect.

### Add Action

A computer screen shot of a program code

AI-generated content may be incorrect.

A black and white rectangle with a white stripe

AI-generated content may be incorrect.

### Remove Action

Next, we will **implement the Remove action** in our WatchlistController.  
This action will **allow logged-in users to remove movies** **from their Watchlist**.

A computer screen shot of text

AI-generated content may be incorrect.

A rectangular object with a white rectangle

AI-generated content may be incorrect.

A yellow rectangle with black text

AI-generated content may be incorrect.

## Workshop III – GitHub Version

### Version Available Online

* The completed solution for **Workshop III** is published and accessible at:

<https://github.com/KTsaneff/ASP.NET-Core-SoftUni-CSharpWeb-May-2025-CinemaApp/tree/Workshop-III>